Project information

- Project name: parallel maximum flow
- Team members: Siyuan Chen, Xinyue Yang
- URL: `https://xinyue-yang.github.io/parallel-maxflow/`

Summary

We are going to parallelize Dinic's maximum flow algorithm under the shared address space model and analyze their performance on GHC and PSC machines.

Progress

- We reviewed several papers on efficient parallel maximum flow algorithms. It turns out that most academic research in this topic is fairly theoretical and focuses primarily on the performance of the algorithm under abstract parallel machine models. These papers are therefore not useful to us as we would like to (and to be honest, are limited by our technical abilities to) focus on improving concrete, real-life performance of maximum flow algorithms.
- We discussed the scope of the project and broke down the code deliverables into basic infrastructure code (network, I/O), sequential implementation of Dinic's, parallel implementation of Dinic's, testing and benchmarking framework, and analysis and visualization code.
- We defined/implemented the following things.
  - Infrastructure: file formats, network representation, I/O, timing
  - Sequential Dinic's
  - Naive test case generation script
- We looked into generating test cases with specific structures, such as those featuring vertices with high degrees and/or long chains of vertices. For the final analysis and benchmarking, we plan to use a few naively-generated networks with various sizes as well as some networks with special structures.
- We began experimenting with parallelzing both the BFS and the DFS components of Dinic's. The BFS portion should be straightforward, although we do anticipate doing quite a bit of testing to tune the performance of the centralized task queue. The DFS seems harder to parallelize, but we have some ideas in terms of pushing through disjoint flows.

Goals

We are mostly adhering to the same goals as outlined in our project proposal. However, due to time constraints we will no longer be parallelizing the alternative maximum flow algorithm (push-relabel).

- Implement sequential Dinic's algorithm.
- Parallelize the first step (BFS) of Dinic's algorithm using parallel set union.
- Parallelize the second step (DFS) of Dinic's algorithm to a reasonable degree using fine-grained locking.
- Profile and analyze sequential and parallel Dinic's algorithm:
  - on different architectures for different core counts: GHC, PSC;
  - for different graph sizes; and
  - for several different graph categories: dense, sparse, high-degree, low-degree, etc.
- Provide a deep understanding of the bottlenecks and weaknesses of the current parallel implementation of Dinic's algorithms.

Poster session

Our deliverables for the poster session are:

- Diagram explaining sequential Dinic's
- Diagram explaining our parallelization of Dinic's, including both the BFS and the DFS
- Performance graphs demonstrating the algorithms' performance on different networks:
  - naively randomly generated test cases of various sizes
  - test cases with special structures

Issues

From our initial experiments running the sequential algorithm on the generated networks, it seems that Dinic's is already extremely fast without parallelization. We are slightly worried that our parallel speedup might potentially suffer due to Amdahl's law, in that the parallelization benefit isn't significant when considering the overhead. For us, it is also difficult to blindly increase the input size as our current networks sizes are already at the order of megabytes, which our sequential implementation takes under a second to run.

Schedule

- Week 4 (Apr. 15th–Apr. 21st):
  - Implement testing framework
  - Parallelize the first step (BFS) of Dinic's algorithm using parallel set union
  - Explore special graph structures that hit bottleneck of sequential algorithm
- Week 5 (Apr. 22nd–Apr. 28th):

  - Parallelize the second step (DFS) of Dinic's algorithm to a reasonable degree using fine-grained locking.
  - Run experiments on different graphs and machines
  - Implement initial data analysis and visualization
  - Design benchmark for parallel algorithm

- Week 6 (Apr. 29th–May. 5th):

  - Finalize data analysis and visualization
  - Make performance graphs demonstrating the algorithms' performance on different networks
  - Write final report and design poster